图文 040、案例实战：每日百亿数据量的实时分析引擎，为啥频繁发生

详情 评论

案例实战：

每日百亿数据量的实时分析引擎，为啥频繁发生Full GC？

给大家推荐一套质量极高的Java面试训练营课程：

作者是中华石杉，石杉老哥是我之前所在团队的 Leader ，骨灰级的技术神牛！

大家可以点击下方链接，了解更多详情，并进行试听：

21天互联网Java进阶面试训练营（分布式篇）

1、上文案例再分析

这个案例将会给大家分析一个频繁Full GC的真实生产案例，我们会延续之前讲过的一个案例，继续进行分析，下面先把之前的案例贴出来放在下文。

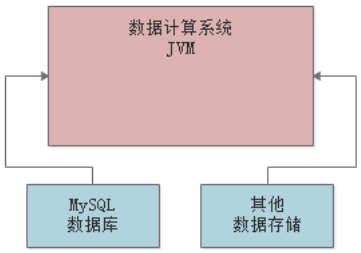
之前分析过的一个日处理上亿数据的计算系统，就是本文所要分析的“每日百亿数据量的实时分析系统”。

2、一个日处理上亿数据的计算系统

先给大家说一下这个系统的案例背景，大概来说是当时我们团队里自己研发的一个数据计算系统，日处理数据量在上亿的规模。

为了方便大家集中注意力理解这个系统的生产环境的JVM相关的东西，所以对系统本身就简化说明了。

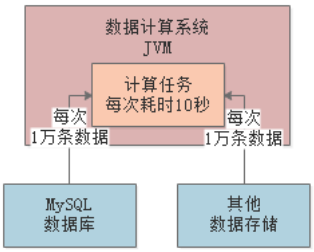
简单来说，这个系统就是会不停的从MySQL数据库以及其他数据源里提取大量的数据加载到自己的JVM内存里来进行计算处理，如下图所示。



这个数据计算系统会不停的通过SQL语句和其他方式，从各种数据存储中提取数据到内存中来进行计算，大致当时的生产负载是每分钟大概需要执行500次数据提取和计算的任务。

但是这是一套分布式运行的系统，所以生产环境部署了多台机器，每台机器大概每分钟负责执行100次数据提取和计算的任务。

每次会提取大概1万条左右的数据到内存里来计算，平均每次计算大概需要耗费10秒左右的时间，然后每台机器是4核8G的配置，JVM内存给了4G，其中新生代和老年代分别是1.5G的内存空间，大家看下图。



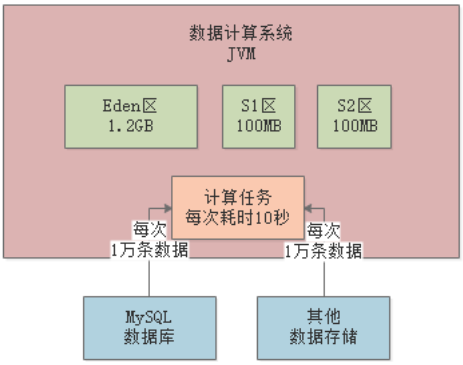
3、这个系统到底多块会塞满新生代?

现在明确了一些核心数据，接着我们来看看这个系统到底多快会塞满新生代的内存空间?

既然这个系统每台机器上部署的实例，每分钟会执行100次数据计算任务，每次是1万条数据需要计算10秒的时间，那么我们来看看每次1万条数据大概会占用多大的内存空间?

这里每条数据都是比较大的，大概每条数据包含了平均20个字段，可以认为平均每条数据在1KB左右的大小。那么每次计算任务的1万条数据就对应了10MB的大小。

所以大家此时可以思考一下，如果新生代是按照8:1:1的比例来分配Eden和两块Survivor的区域，那么大体上来说，Eden区就是1.2GB，每块Survivor区域在100MB左右，如下图。



基本上按照这个内存大小而言，大家会发现，每次执行一个计算任务，就会在Eden区里分配10MB左右的对象，那么一分钟大概对应100次计算任务，其实基本上一分钟过后，Eden区里就全是对象，基本就全满了。

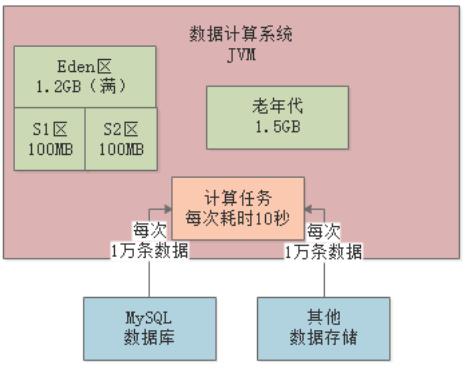
所以说，回答这个小节的问题，新生代里的Eden区，基本上1分钟左右就迅速填满了。

4、触发Minor GC的时候会有多少对象进入老年代?

此时假设新生代的Eden区在1分钟过后都塞满对象了，然后在接着继续执行计算任务的时候，势必会导致需要进行Minor GC回收一部分的垃圾对象。

那么上篇文章给大家讲过这里在执行Minor GC之前会先进行的检查。

首先第一步，先看看老年代的可用内存空间是否大于新生代全部对象？看下图，此时老年代是空的，大概有1.5G的可用内存空间，新生代的Eden区大概算他有1.2G的对象好了。

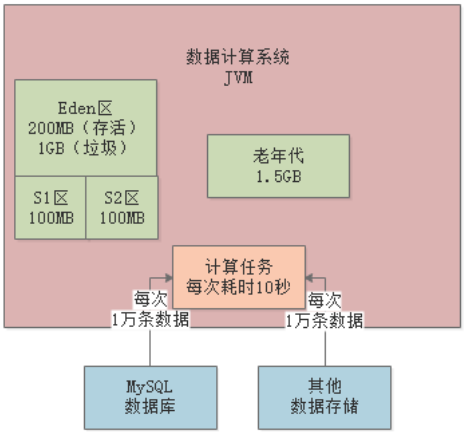


此时会发现老年代的可用内存空间有1.5GB，新生代的对象总共有1.2GB，即使一次Minor GC过后，全部对象都存活，老年代也能放的下的，那么此时就会直接执行Minor GC了。

那么此时Eden区里有多少对象还是存活的，无法被垃圾回收呢?

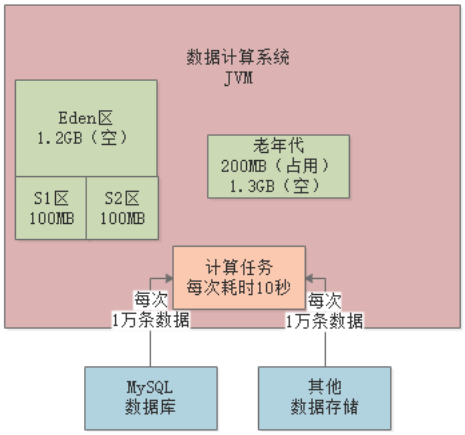
大家可以考虑一下之前说的那个点，每个计算任务1万条数据需要计算10秒钟，所以假设此时80个计算任务都执行结束了，但是还有20个计算任务共计200MB的数据还在计算中

那么此时就是200MB的对象是存活的，不能被垃圾回收掉，然后有1GB的对象是可以垃圾回收的，大家看下图。



此时一次Minor GC就会回收掉1GB的对象，然后200MB的对象能放入Survivor区吗?

不能！ 因为任何一块Survivor区实际上就100MB的空间，此时就会通过空间担保机制，让这200MB对象直接进入老年代去，占用里面200MB内存空间，然后Eden区就清空了，大家看下图。



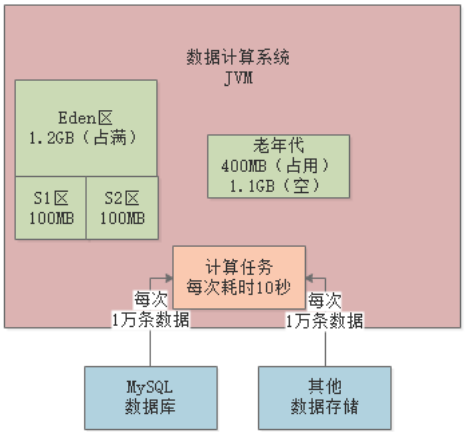
5、系统运行多久，老年代大概就会填满?

那么大家想一下，这个系统大概运行多久，老年代会填满呢?

按照上述计算，每分钟都是一个轮回，大概算下来是每分钟都会把新生代的Eden区填满，然后触发一次Minor GC，然后大概都会有200MB左右的数据进入老年代。

那么大家可以想一下，假设现在2分钟运行过去了，此时老年代已经有400MB内存被占用了，只有1.1GB的内存可用，此时如果第3分钟运行完毕，又要进行Minor GC会做什么检查呢?

如下图:



此时会先检查老年代可用空间是否大于新生代全部对象！

此时老年代可用空间1.1GB，新生代对象有1.2GB，那么此时假设一次Minor GC过后新生代对象全部存活，老年代是放不下的，那么此时就得看看一个参数是否打开了 。

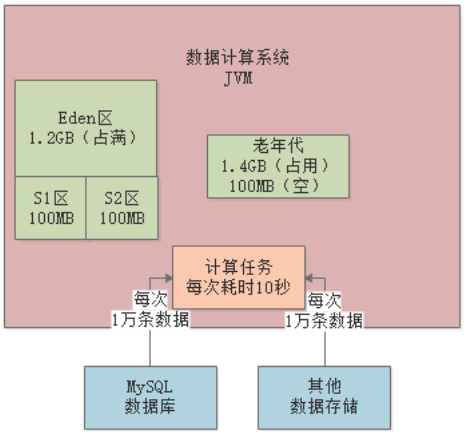
如果“-XX:-HandlePromotionFailure”参数被打开了，当然一般都会打开其实，此时会进入第二步检查，就是看看老年代可用空间是否大于历次Minor GC过后进入老年代的对象的平均大小。

我们已经计算过了，大概每分钟会执行一次Minor GC，每次大概200MB对象会进入老年代。

那么此时发现老年代的1.1GB空间，是大于每次Minor GC后平均200MB对象进入老年代的大小的，所以基本可以推测，本次Minor GC后大概率还是有200MB对象进入老年代，1.1G可用空间是足够的。

所以此时就会放心执行一次Minor GC，然后又是200MB对象进入老年代。

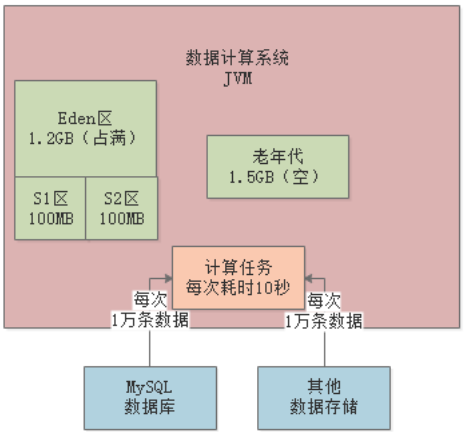
转折点大概在运行了7分钟过后，7次Minor GC执行过后，大概1.4G对象进入老年代，老年代剩余空间就不到100MB了，几乎快满了，如下图。



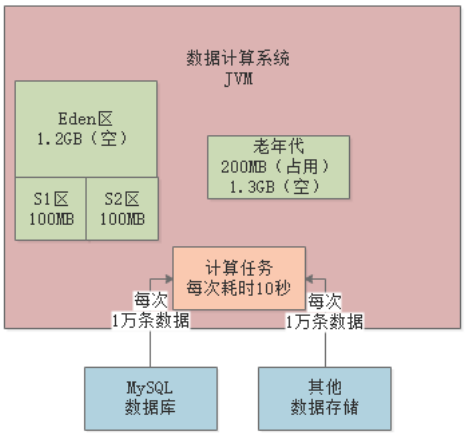
6、这个系统运行多久，老年代会触发1次Full GC?

大概在第8分钟运行结束的时候，新生代又满了，执行Minor GC之前进行检查，此时发现老年代只有100MB内存空间了，比之前每次Minor GC后进入老年代的200MB对象要小，此时就会直接触发一次Full GC。

Full GC会把老年代的垃圾对象都给回收了，假设此时老年代被占据的1.4G空间里，全部都是可以回收的对象，那么此时一次性就会把这些对象都给回收了，如下图。



然后接着就会执行Minor GC，此时Eden区情况，200MB对象再次进入老年代，之前的Full GC就是为这些新生代本次Minor GC要进入老年代的对象准备的，如下图。



按照这个运行模型，基本上平均就是七八分钟一次Full GC，这个频率就相当高了。

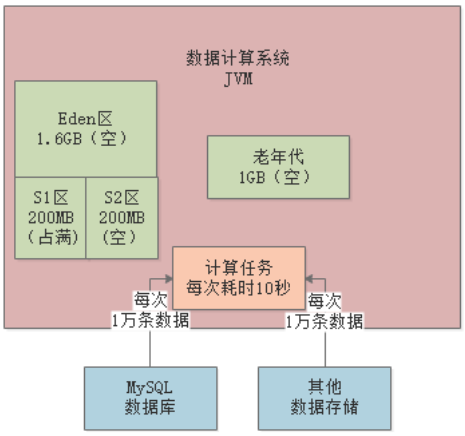
因为每次Full GC速度都是很慢的，性能很差，而且**明天的文章会告诉大家，为什么Full GC的时候会严重影响系统性能。**

7、该案例应该如何进行JVM优化?

相信通过这个案例，大家结合图一路看下来，对新生代和老年代如何配合使用，然后什么情况下触发Minor GC和Full GC，什么情况下会导致频繁的Minor GC和Full GC，大家都有了更加深层次和透彻的理解了。

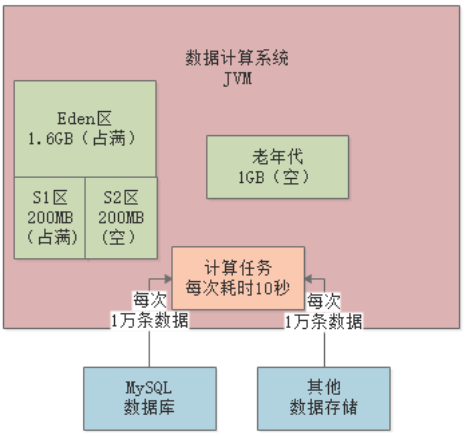
对这个系统，其实要优化也是很简单的，因为这个系统是数据计算系统，每次Minor GC的时候，必然会有一批数据没计算完毕，但是按照现有的内存模型，最大的问题，其实就是每次Survivor区域放不下存活对象。

所以当时我们就是对生产系统进行了调整，增加了新生代的内存比例，3GB左右的堆内存，其中2GB分配给新生代，1GB留给老年代，这样Survivor区大概就是200MB，每次刚好能放得下Minor GC过后存活的对象了，如下图所示。

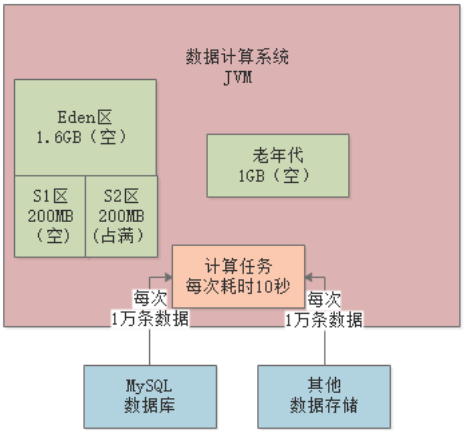


只要每次Minor GC过后200MB存活对象可以放Survivor区域，那么等下一次Minor GC的时候，这个Survivor区的对象对应的计算任务早就结束了，都是可以回收的

此时比如Eden区里1.6GB空间被占满了，然后Survivor1区里有200MB上一轮 Minor GC后存活的对象，如下图。



然后此时执行Minor GC，就会把Eden区里1.4GB对象回收掉，Survivor1区里的200MB对象也会回收掉，然后Eden区里剩余的200MB存活对象会放入Survivor2区里，如下图。



以此类推，基本上就很少对象会进入老年代中，老年代里的对象也不会太多的。

通过这个分析和优化，定时我们成功的把生产系统的老年代Full GC的频率从几分钟一次降低到了几个小时一次，大幅度提升了系统的性能，避免了频繁Full GC对系统运行的影响。

8、如果该系统的工作负载再次扩大10倍呢?

相信大家之前都看过这个案例了，这次正好借着这个机会再次重看一遍，加深一下印象，同时我们接着说当时那个生产系统在每日处理1亿数据之后，随着一段时间过后，工作负载再次扩大10倍的情景。

如果工作负载扩大10倍，那么大家参照上图来看，此时会导致每秒钟要加载100MB的数据到内存里去，对于1.6G的Eden而言，10多秒就会迅速塞满，此时就会触发Young GC。

但是之前说过，你每次加载一批数据到内存里去，一般要处理10秒以上的时间才能计算完毕，在计算完毕之前这些数据是不能被回收的。

所以如果你10多秒就触发一次Young GC，直接导致的后果就是，此时可能能回收掉的垃圾也就几百MB而言，可能1GB的对象都是无法回收的，大家仔细理解一下这个意思。

此时就会导致每隔10多秒，就有1GB的数据进入老年代中，而老年代之前给大家说过，也就1GB左右的空间而已，即使勉强让你放下了，那么下一次过10多秒之后，又会放1GB的对象到老年代，此时必然会提前触发Full GC去回收老年代里的1GB的对象，然后再让你把这次Young GC后存活的1GB对象放入老年代。

这就是当时我们遇到的真实生产场景，基本上一台4核8G的机器，每分钟要触发二三次Full GC，对系统性能造成了巨大的影响，简直是可怕至极。

9、使用大内存机器来优化上述场景

所以但是针对这个问题，因为考虑是计算类的系统，也是非常的吃内存的，所以同样是更换成了每台机器都是16核32G的高配置机器

这样的话，Eden基本上空间会扩大10倍，比如有16GB。

那么此时按照每秒加载100MB的数据到内存里进行计算，要2分钟左右才会触发一次Young GC，因为降低了Young GC的频率，所以每次Young GC的时候存活对象大概也就几百MB而已，不会超过1GB。

当时给Survivor区域分配的是每个Survivor有2GB内存，所以每次Young GC过后的存活对象可以轻松放入Survivor区域中，不会进入老年代。这就完美的通过提升机器配置的方式，解决了频繁Young GC和Full GC的问题。

很多同学可能会提问了，那么针对大内存机器，需要用G1来减少每次Young GC的停顿时间吗?

答案是：不用。因为这是一个后台自动进行计算的系统，他不是直接面向用户的系统，所以哪怕每隔2分钟一次Young GC，一次要停顿1秒钟，也对系统几乎没任何影响。

10、本文总结

这篇文章接着之前的案例，让大家看了一下，1亿数据量级下的系统部署4核8G的机器，Full GC为何频繁发生，如何优化？10亿量级下的系统部署在4核8G的机器上，Full GC会发生的有多么的恐怖，如何通过提升机器配置来优化？

相信大家仔细看完这个案例，多看几遍，透彻理解了，多频繁Full GC问题就彻底理解了。

11、小小思考题

今日给大家留的一个小小思考题：

看看你们线上系统一般每隔多长时间发生一次Full GC？
每次Full GC持续多久？
对你们系统的性能有影响吗？

End

常见问题解答：

一、 如何生成自己的分享海报并获取返现？

方式1：

点击文章右上角**邀请好友**（如下图），生成自己的专属海报。

将海报发送给好友或分享朋友圈，朋友通过扫描你分享的海报购买课程，你将**获取返现24元**，可在个人中心中提现：